**UVVM Essential Mechanisms** –Quick Reference  
  
This document explains some of the essential mechanisms necessary for running UVVM, in addition to helpful and important VVC status and configuration records which are accessible directly from the testbench. A more comprehensive review can be found in the VVC Framework Manual.

[1 Libraries 1](#_Toc17902587)

[2 UVVM Initialization 2](#_Toc17902588)

[3 UVVM and VVC Shared Variables 3](#_Toc17902589)

[4 VVC Status, Configuration and Transaction information 4](#_Toc17902590)

[5 Direct Transaction Transfer 4](#_Toc17902591)

[5.1 Transaction definitions 4](#_Toc17902592)

[5.2 Transaction details 4](#_Toc17902593)

[6 Multiple Central Sequencers 4](#_Toc17902594)

[7 Compile scripts 5](#_Toc17902595)

[8 Scope of verbosity control 5](#_Toc17902596)



# Libraries

In order to use a VVC the following libraries need to be included:

library uvvm\_util;

context uvvm\_util.uvvm\_util\_context;

library uvvm\_vvc\_framework;

use uvvm\_vvc\_framework.ti\_vvc\_framework\_support\_pkg.all;

library bitvis\_vip\_<name>;

context bitvis\_vip\_<name>.vvc\_context;

# UVVM Initialization

The following mechanisms are required for running UVVM

|  |  |
| --- | --- |
| **Mechanism** | **Description** |
| **ti\_uvvm\_engine** | **ti\_uvvm\_engine**  This entity contains a process that will initialize the UVVM environment, and has to be instantiated in the testbench harness, or alternatively in the top-level testbench.  Example:  i\_ti\_uvvm\_engine : entity uvvm\_vvc\_framework.ti\_uvvm\_engine; |
| **await\_uvvm\_initialization()** | **await\_uvvm\_initialization(VOID)**  This procedure is a blocking procedure that has to be called from the testbench sequencer, prior to any VVC calls, to ensure that the UVVM engine has been initialized and is ready. This procedure will check the shared\_uvvm\_state on each delta cycle until the UVVM engine has been initialized.  Note that this method is depending on the ti\_uvvm\_engine mechanism.  Note that this method uses the t\_void parameter, defined in the UVVM Utility Library types package.  Example:  await\_uvvm\_initialization(VOID); |

# UVVM and VVC Shared Variables

UVVM and VVC shared variables are defined in global\_signals\_and\_shared\_variables\_pkg and the various vvc\_methods\_pkg, respectively.

Shared variables

|  |  |
| --- | --- |
| **Shared variable** | **Description** |
| shared\_uvvm\_status | Shared variable providing access to VVC related information via the info\_on\_finishing\_await\_any\_completion record element, i.e.  shared\_uvvm\_status.info\_on\_finishing\_await\_any\_completion  This record element gives access to the name, command index and the time of completion of the VVC that first fulfilled the await\_any\_completion(). The available record fields are:  vvc\_name : string -- default “no await\_any\_completion() yet”  vvc\_cmd\_idx : natural -- default 0  vvc\_time\_of\_completion : time -- default 0 ns  For more information regarding other fields available in the shared\_uvvm\_status see the UVVM Util QuickRef, section 1.4 |
| shared\_<vvc\_name>\_vvc\_config | Shared variable providing access to configuration parameters for each VVC instance and channel if applicable. E.g.  shared\_sbi\_vvc\_config(1).inter\_bfm\_delay.delay\_type := TIME\_START2START;  shared\_uart\_vvc\_config(RX,1).bfm\_config.bit\_time := C\_BIT\_TIME; |
| shared\_<vvc\_name>\_vvc\_status | Shared variable providing access to status parameters for each VVC instance and channel if applicable. E.g.  v\_num\_pending\_cmds := shared\_sbi\_vvc\_status(1).pending\_cmd\_cnt;  v\_current\_cmd\_idx := shared\_uart\_vvc\_status(TX,2).current\_cmd\_idx;  v\_previous\_cmd\_idx := shared\_uart\_vvc\_status(TX,2).previous\_cmd\_idx; |
| shared\_<vvc\_name>\_transaction\_info | Shared variable providing access to VVC instances transaction information to include in the wave view during simulation.  Available information is dependent on VVC type and typical information is:  operation : t\_operation; -- default NO\_OPERATION  data : std\_logic\_vector(C\_VVC\_CMD\_DATA\_MAX\_LENGTH-1 downto 0); -- default 0x0  msg : string(1 to C\_VVC\_CMD\_STRING\_MAX\_LENGTH); -- default empty |

# VVC Status, Configuration and Transaction information

The VVC status, configuration and transaction information records are defined in each individual VVC methods package.

Each VVC instance and channel can be configured and useful information can be accessed from the testbench via dedicated shared variables.

From the VVC configuration shared variable, one is given the ability to tailor each VVC to one’s needs, in addition to access the BFM configuration record via the bfm\_config identifier. In addition to BFM configuration possibility, the configuration settings consist of command and result queue settings, BFM access separation delay and a VVC dedicated message ID panel.  
Note that some BFMs require user configuration, e.g. the bit\_time setting in serial interface BFMs.

The VVC status shared variable provide access to the command status parameters for each of the VVCs, such as the current and previous command index, and the number of pending commands in the VVCs command queue. This provide a helpful tool, e.g. when synchronizing VVCs in the test sequencer using the await\_completion() or await\_any\_completion() methods.

When using a wave viewer during simulation, the transaction shared variable provides helpful information regarding current VVC operation and transaction information such as address and data. Note that the accessible fields depend on the VVC and its implementation. An example of two SBI VVCs performing FIFO write operations, followed by check operations, is shown in Figure 4‑1.



Figure 4‑1 VVC Transaction info example

# Direct Transaction Transfer

From the release of UVVM version 3.0 the testing framework can now benefit from an even more advanced transaction model, allowing sharing of transaction information among testbench sequencer, VVCs, monitors, models and scoreboards.

Direct transaction transfer in UVVM is a way of transferring all required information about a transaction on the DUT interface out of the VVC at a high level of abstraction. This is a powerful mechanism that allow the model, sequencer or other part of the testbench to get information about a DUT access without needing to analyse the actual interface.

## Transaction definitions

The type of transaction depends on its origin and destination, and with UVVM Direct Transaction Transfer (DTT) the following transaction terms applies:

* Base transaction (BT) is the lowest level of a complete transaction as allowed from the central sequencer. E.g. Avalon\_Read()
* Sub-transaction (ST) is the lowest level of an incomplete transaction as allowed from a BFM. E.g. Avalon\_Read\_Request()
* Leaf transaction (LT) is the lowest level of complete or incomplete transaction. I.e. a sub-transaction when this is defined, otherwise a base transaction.
* Compound transaction (CT) is a set of transactions or others methods or statements that as a total is doing a more complex operation. E.g. SBI\_Poll\_Until().

## Transaction details

The transaction information medium is a global signal that hold all required information gathered in a transaction record.

Note that transaction elements may vary, depending on type of VVC, interface and testbench environment. In Table 1 and Table 2 the required VVC record elements are listed in white table rows, while grey table rows show potential record elements, and Table 3 lists Monitor record elements.

Note that a VVC will update its DTT details prior to transaction request, i.e. one delta cycle before requesting BFM activity, while a monitor will set its DTT information after BFM activity (LT) has finished, and that all DTT information is set back to default values activity has finished.

Table 1 - VVC Transaction record t\_vvc\_transaction – not applicable for Monitors

|  |  |  |
| --- | --- | --- |
| **Record element** | **Type** | **Description** |
| bt | t\_transaction | Base transaction, as described in 5.1 |
| ct | t\_transaction | Compound transaction, as described in 5.1 |

Table 2 – VVC transaction record t\_transaction

|  |  |  |  |
| --- | --- | --- | --- |
| **Record element** | **Type** | **Default** | **Description** |
| Operation | t\_operation | NO\_OPERATION | Current state of operation on DUT. E.g. NO\_OPERATION, WRITE, READ, POLL\_UNTIL.  All operations will be separated with a NO\_OPERATION for at least 1 delta cycle, e.g. NO\_OPERATION – WRITE – NO\_OPERATION – READ – NO\_OPERATION. |
| Address | unsigned | 0x0 | Requested DUT address to read / write |
| Data | std\_logic\_vector | 0x0 | Requested DUT data to write / expect |
| Transaction\_validity | t\_transaction\_validity | VALID | The transaction intention as seen from the VVC. **1**  Supported validity types are VALID, INVALID, INTENDED\_VALID, INTENDED\_INVALID |
| Meta | t\_vvc\_meta | msg = “”, cmd\_idx = -1 | Information essential to VVCs, i.e. VVC command index and message. |
| Error\_info | t\_error\_info | false | Any type of error injection implemented in a DUT access, e.g. parity error |

Table 3 - Monitor transaction record t\_transaction

|  |  |  |  |
| --- | --- | --- | --- |
| **Record element** | **Type** | **Default** | **Description** |
| Operation | t\_operation | NO\_OPERATION | Current state of operation on DUT. E.g. NO\_OPERATION, WRITE, READ, POLL\_UNTIL.  All operations will be separated with a NO\_OPERATION for at least 1 delta cycle, e.g. NO\_OPERATION – WRITE – NO\_OPERATION – READ – NO\_OPERATION. |
| Address | unsigned | 0x0 | Requested DUT address to read / write |
| Data | std\_logic\_vector | 0x0 | Requested DUT data to write / expect |
| Transaction\_validity | t\_transaction\_validity | VALID | The transaction intention as seen from the VVC. **1** Supported validity types are VALID, INVALID |
| Meta | t\_vvc\_meta | msg = “”, cmd\_idx = -1 | Information essential to VVCs. In a monitor these fields will always have default values: msg = ”” and cmd\_idx = -1. |
| Error\_info | t\_error\_info | false | Any type of error injection implemented in a DUT access, e.g. parity error |

Note 1: the VVC do not know the success of a BFM accessing a DUT, thus the VVC can only report its intended transaction details.

Note 2: The Monitor know the success of a DUT transaction, thus it can set the transaction\_validity to VALID or INVALID.

# Multiple Central Sequencers

A structured test environment is important and we recommend the use of a test harness to instantiate VVCs, DUT, clock generator and so forth. The testbench may consist of one or more test sequencers which are used to control the complete testbench architecture with any number of VVCs, although for a better testbench overview we recommend to have a single central test sequencer only.

# Compile scripts

In the script folder in the root directory the compile\_all.do compiles all UVVM components. This script may be called with one to three input arguments. The first input argument is the directory of the script folder at the root directory from the working directory. The second input argument is the target directory of the compiled libraries, by default every library is compiled in a sim folder in the corresponding components directory. The third input argument is the directory to a custom component list in .txt format. The script will only compile the components listed in that file. By default, the script uses the file component\_list.txt located in uvvm/script. This file can be modified so that only some components are compiled.

Example: do uvvm/script/compile\_all.do uvvm/script

There are also compile scripts for all UVVM components located in the script folder of each UVVM component. These scripts can be called with two input arguments. The first input argument is the directory of the component folder from the working directory. The second input argument is the target directory of the compiled library, default is the sim folder in the respective component.

Example: do uvvm/uvvm\_util/script/compile\_src.do uvvm/uvvm\_util

# Scope of verbosity control

Message IDs are used for verbosity control in many of the procedures and functions in UVVM, as well as log messages and checks in VVCs, BFMs and Scoreboards.   
Note that VVCs and Scoreboards comes with dedicated message ID panels and are not affected by the global message ID panel, but accessed by addressing targeting VVC or Scoreboard and, if applicable, instance number or with a broadcast.

E.g.

-- Global message ID panel. Does not apply to VVCs or Scoreboards, as they have their own local message ID panel

disable\_log\_msg(ALL\_MESSAGES);

enable\_log\_msg(ID\_SEQUENCER);

-- VVC message ID panel  
disable\_log\_msg(VVC\_BROADCAST, ALL\_MESSAGES); -- broadcast to all VVCs and instances

enable\_ log\_msg(I2C\_VVCT, C\_VVC\_INSTANCE\_1, ID\_BFM\_WAIT); -- I2C VVC instance 1  
enable\_log\_msg(I2C\_VVTC, C\_VVC\_INSTANCE\_2, ID\_BFM\_WAIT); -- I2C VVC instance 2

-- Scoreboard message ID panel

shared variable sb\_under\_test : record\_sb\_pkg.t\_generic\_sb;

…

sb\_under\_test.disable\_log\_msg(ALL\_INSTANCES, ID\_CTRL); -- broadcast to all SB instances

sb\_under\_test.enable\_log\_msg(C\_SB\_INSTANCE\_1, ID\_DATA); -- SB instance 1

A subset of message IDs is listed in UVVM Utility Library QR, section 1.10.

Disclaimer: This IP and any part thereof are provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement.  
In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with this IP.

**INTELLECTUAL**

**PROPERTY**